A COMPARISON OF LOGICAL AND PHYSICAL PARALLEL I/O PATTERNS

Huseyin Simitci
Daniel A. Reed

DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN, URBANA, ILLINOIS, U.S.A.

Summary

Although there are several extant studies of parallel scientific application request patterns, there is little experimental data on the correlation of physical I/O patterns with application I/O stimuli. To understand these correlations, the authors have instrumented the SCSI device drivers of the Intel Paragon OSF/1 operating system to record key physical I/O activities, and have correlated this data with the I/O patterns of scientific applications captured via the Pablo analysis toolkit. This analysis shows that disk hardware features profoundly affect the distribution of request delays and that current parallel file systems respond to parallel application I/O patterns in nonscalable ways.

Address reprint requests to Huseyin Simitci, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 West Springfield Avenue, Urbana, IL 61801 U.S.A., e-mail simitci@cs.uiuc.edu.

The International Journal of High Performance Computing Applications, Volume 12, No. 3, Fall 1998, pp. 364-380

1 Introduction

I/O for scalable parallel systems continues to be the major performance bottleneck for many large-scale scientific applications (Crandall et al., 1996; Purakayastha et al., 1995). Market forces are increasing the disparity between processor and disk system performance, exacerbating the already difficult problem of achieving high performance for applications with large I/O components. Moreover, most current parallel file systems (PFS) were constructed as extensions of workstation file systems and were optimized for large sequential data transfers.

Recent experimental studies (Crandall et al., 1996; Purakayastha et al., 1995; Smirni and Reed, 1996, 1997; Reed, Elford, Madhyastha, Scullin, et al., 1996) have shown that parallel applications have much more complex access patterns, with greater spatial and temporal variability, than first suspected. Although there is a large, complementary body of experimental data on disk behavior (Ruemmler and Wilkes, 1993, 1994) for sequential file systems, there is much less experimental data on the correlation of physical I/O patterns with parallel application I/O stimuli. Because PFSs mediate application I/O stimuli and physical I/O responses, developing appropriate designs for scalable parallel I/O systems requires a detailed characterization of the I/O behavior at multiple system levels (Gibson, Vitter, and Wilkes, 1996).

To correlate parallel application I/O stimuli with disk system responses, we augmented our portable application I/O instrumentation infrastructure (Reed, Elford, Madhyastha, Scullin, et al., 1996) with disk device driver instrumentation. Built atop the Pablo performance analysis toolkit, the former can capture both statistical summaries and time-stamped traces of application I/O patterns. In turn, our device driver instrumentation creates temporal summaries and activity histograms for selected SCSI devices, including read/write request sizes, device driver delays, SCSI device service time, response times, and queue lengths. Using this experimental infrastructure, we have studied logical and physical I/O patterns for three disk configurations of the Intel Paragon™ XP/S and the Intel PFS, one of the few extant commercial PFSs now available.

The remainder of this paper is organized as follows. In Section 2, we outline related work in parallel I/O characterization and disk modeling. We then describe our logical (application) and physical (disk) I/O characterization methodology in Section 3. As a baseline for analysis of I/O behavior in large scientific applications, Section 4

summarizes logical and physical I/O characteristics for a set of simple benchmarks. This is followed in Section 5 by a description of MESSKIT (High Performance Computational Chemistry Group, 1995), a large, multiphase quantum chemistry code with demanding I/O requirements that are representative of current parallel scientific applications. In Sections 6 and 7, we analyze the logical and physical I/O patterns for MESSKIT when executed on three different hardware configurations. Finally, Sections 8 and 9 summarize our findings and outline directions for future work.

2 Related Work

Although our understanding of I/O parallelism is still evolving, there is a long history of file access characterization for mainframes and vector supercomputers. Notable examples include Lawrie, Randal, and Barton's (1982) study of automatic file migration algorithms, Stritter's (1977) analysis of file lifetime distributions, Smith's (1981) study of mainframe file access behavior, and Jensen and Reed's (1993) study of file archive accesses.

More recently, Miller and Katz (1991) captured detailed traces of application file accesses from a suite of Cray applications, identifying compulsory, checkpoint, and staging I/O. Pasquale and Polyzos (1993, 1994) followed with two additional studies of vector workloads, concluding that most I/O had regular behavior.

In the parallel domain, Kotz and colleagues (Kotz and Nieuwejaar, 1994; Purakayastha et al., 1995) used library instrumentation to study I/O patterns on the Intel iPSC/860 and the Thinking Machines CM-5. They observed that parallel I/O patterns were more complex than expected, with small requests quite common.

Complementary studies of physical I/O patterns have focused on modeling and simulation of single disks (Ruemmler and Wilkes, 1994) and analysis of disk workloads in UNIX systems (Ruemmler and Wilkes, 1993; Baker, 1991). This work showed that without higher level file system optimizations, the benefits from even the best disk scheduling algorithms were limited (Seltzer, Chen, and Ousterhout, 1990).

Our work differs from these earlier studies by examining the correlations between parallel application I/O requests, PFS policies, and physical disk request streams. This correlation is a prerequisite to understanding how PFSs mediate and transduce logical and physical request streams and can provide a basis for intelligent design of massively PFSs.

"Because PFSs mediate application I/O stimuli and physical I/O responses, developing appropriate designs for scalable parallel I/O systems requires a detailed characterization of the I/O behavior at multiple system levels."

3 Experimental Methodology

Application requests are the logical stimuli to an I/O system; their sizes, temporal spacing, and spatial patterns (e.g., sequential or random) constrain possible library and file system optimizations (e.g., by prefetching or caching). After mediation by a PFS, the physical patterns of I/O manifest at the storage devices are the ultimate system response.

To understand the implications of application request patterns for PFSs and the efficacy of parallel disk configurations, we have augmented the Pablo performance analysis environment's support for application I/O tracing (Crandall et al., 1996) with SCSI device driver instrumentation on the Intel Paragon XP/S system. Below, we describe the experimental platform and our measurement toolkit.

3.1 EXPERIMENTAL PLATFORM

The distributed memory Paragon XP/S architecture consists of a group of compute and I/O nodes, all connected by a two-dimensional mesh. These nodes execute a distributed version of OSF/1 AD, with application I/O requests on the compute nodes sent to file servers on the I/O nodes. In turn, the I/O nodes support Intel's PFS. The PFS stripes files across the I/O nodes in 64-KB units, with the initial 64-KB file segment randomly placed on one of the I/O nodes; subsequent 64-KB stripes are distributed using a round-robin algorithm.

Together, OSF/1 and the PFS I/O node file servers support both buffered and unbuffered modes, selectable via a system call for each file. When buffering is enabled, PFS and OSF/1 use a read-ahead and write-behind caching algorithm with LRU replacement and 64-KB units. Finally, if the last two consecutive disk reads have contiguous logical block numbers, one more logical file block is prefetched asynchronously.

By default, PFS buffering is disabled, and a technique called fast path I/O is used to avoid data caching and copying. In this case, I/O node buffer caches and client side memory mapped file support are bypassed, and data are transferred directly between disks and user buffers.

All our experiments were conducted on the 512-node Intel Paragon XP/S at the Center for Advanced Computing Research at the California Institute of Technology (Caltech) and used OSF/1 version R1.4.1. As a major test platform for the Scalable I/O Initiative (Pool, 1996), this system supports multiple I/O configurations, each differing in the number of I/O nodes and both number and type of disks.

In our experiments, we considered two disk types and three different numbers of I/O nodes, each with either a single attached SCSI disk or a single RAID-3 disk array. Disk hardware parameters for each I/O node configuration are given in Table 1. The stripe group is the number of disks on which the files are striped for each I/O hardware configuration, and the stripe size is the amount of data stored on each disk before moving to the next disk. Although the two disk types differ in logical disk block size, the basic unit of storage on the disk, OSF/1 treats the different disks identically; it always transfers data in multiples of a single sector (i.e., multiples of 2 KB) in response to read or write requests.

Varying the number of I/O nodes allowed us to assess the effects of I/O node parallelism on disk I/O patterns and application I/O response times. Conversely, varying the hardware characteristics of the disks allows us to understand the effects of disk capabilities on observed behavior.

Although exploring an even larger set of hardware/software configurations would be desirable, we were constrained by the fact that the Caltech Intel Paragon XP/S is in production use. In consequence, wholesale hardware configuration changes (e.g., modifying the placement and number of disks) or major operating system changes were not practical.

3.2 LOGICAL I/O INSTRUMENTATION

The Scalable I/O Initiative (Pool, 1996) is a broad-based multiagency research group working in concert with vendors to design parallel I/O APIs and file systems. As part of the Scalable I/O Initiative's I/O characterization effort, we have extended the Pablo performance environment (Reed et al., 1993; Reed, Elford, Madhyastha, Scullin, et al., 1996) to capture application I/O behavior on a variety of single processor and parallel systems. The extended Pablo I/O toolkit wraps invocations of I/O routines with instrumentation calls that record the parameters and duration of each invocation.

To minimize potential I/O perturbations due to performance data extraction, the Pablo toolkit supports both real-time reduction of I/O performance data and capture of detailed event traces. These two options trade computation perturbation for I/O perturbation. Extensive use of the Pablo toolkit for application I/O characterization (Crandall et al., 1996; Smirni et al., 1996; Reed, Elford, Madhyastha, Scullin, et al., 1996) has shown that the instrumentation overhead is negligible for most application codes.

3.3 PHYSICAL I/O INSTRUMENTATION

Device drivers define the interface between file system services and I/O devices, isolating the idiosyncrasies of specific devices behind standard interfaces. Because all physical I/O requests transit the device drivers, instrumenting these device drivers allows one to capture and analyze the temporal and spatial patterns of all requests generated by a PFS.

In most current parallel I/O systems, including the Intel Paragon XP/S, the disks are connected to the I/O nodes via standard SCSI controllers, and SCSI device drivers service all requests. A SCSI disk appears externally as a linear vector of addressable blocks. All physical characteristics (i.e., cylinders, tracks, sectors, and bad blocks) are hidden by this virtual device interface.

This separation of logical and physical views simplifies the device driver interface and allows the storage device to transparently optimize requests. However, external entities, including the device driver, have little or no knowledge of the data layout on the physical media; the status of the on-board disk cache or scheduling algorithm; or the seek, rotational latency, or transfer components of a request service time.

Although there are experimental techniques for determining these features (Wilkes et al., 1995), for simplicity's sake we have restricted our analysis to behavior observable at the SCSI device drivers. As a further compromise between detail and overhead, we have opted to generate activity histograms that summarize read and write request sizes, device driver delays, device service times, request run lengths, driver and device queue lengths, and interarrival times.

To generate these histograms, we have modified the SCSI disk driver read/write routines (Forin, Golub, and Bershad, 1991) to time stamp each request on arrival, transmission to or receipt from the device, and departure. Using these time stamps, we then compute the time each request spent queued for service at the device driver, the time spent on the device, and the total response time (i.e., the sum of queuing and service delay). This and other data are kept in a kernel data structure associated with each SCSI device.

Figure 1 illustrates the high-level structure of the resulting SCSI device driver instrumentation. On the Paragon XP/S, processors in the service partition provide general operating system services, and the external interface to the users. Parallel applications are executed on the processors in the compute partition. I/O nodes differ from

Table 1
SCSI Disk Configurations (Intel Paragon XP/S)

Parameter	64 Disks	16 Disks	12 RAIDs
Stripe group	64	16	12
Stripe size	64 KB	64 KB	64 KB
Disk type	Seagate	Seagate	NCR-Maxtor
Interface version	SCSI-2	SCSI-2	SCSI-2
Disk capacity	4 GB	4 GB	2 GB
Logical disk block size	512 bytes	512 bytes	2 KB
Disk sector size	2 KB	2 KB	2 KB
RPM	7200	7200	6300
RAID level	N/A	N/A	3

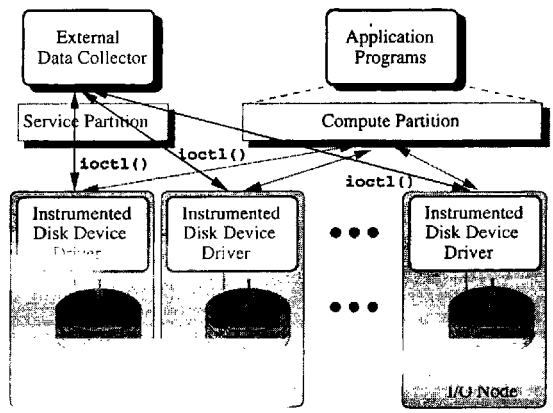


Fig. 1 Physical I/O Instrumentation

Table 2
Benchmark Logical/Physical I/O Comparison (Intel Paragon XP/S)

	Application		64 Disks		16 Disks		12 RAIDs	
Operation	Count	Volume (MB)	Count	Volume (MB)	Count	Volume (MB)	Count	Volume (MB)
Write benchmark (unbuffered)								
`read`	0	0	4941	278.68	5540	346.25	5363	326.20
`write`	8320	650.00	28,830	1384.35	19,776	971.16	20,887	1038.58
Read benchmark (unbuffered)								
`read`	8320	650.00	16,640	650.00	16,704	650.12	17,874	726.84
`write`	0	0	1158	72.37	342	21.04	87	5.27
Write benchmark (buffered)								
`read`	0	0	1389	37.37	264	15.18	918	42.81
`write`	8320	650.00	20,685	1140.43	13,747	812.83	15,557	922.32
Read benchmark (buffered)								
`read`	8320	650.00	13,020	809.75	19,076	1184.38	24,538	1532.99
`write`	0	0	1135	70.93	348	21.42	93	5.48

NOTE: Request size = 80 KB, interrequest time = 1000 ms, file type = private, requests/processor = 130, number of processors = 64, access pattern = sequential.

other compute nodes by having an SCSI interface and an SCSI disk.

During application program execution, an external user program executing on a service node can configure, reset, and retrieve the physical I/O data via an extended set of `ioctl()` calls. When synchronized with application instrumentation, this periodic histogram extraction provides the data needed to correlate logical and physical request patterns.

4 Physical I/O Benchmarks

As a basis for understanding the more complex I/O patterns found in parallel applications (e.g., the quantum chemistry code described in Section 5), we first measured the logical and physical I/O behavior of a suite of benchmarks. Via these configurable benchmarks, one can specify request types and sizes, interrequest latencies, parallelism, file sharing, and PFS buffering.

Table 2 summarizes the logical and physical I/O attributes of two such benchmarks where every processor sequentially reads or writes 80-KB units to or from a private file.[1] In turn, Figures 2 and 3 illustrate the temporal patterns of physical I/O for file read benchmark. The histograms in Figures 2 and 3 are captured and plotted with 10-s intervals, while Figures 4 through 6 are plotted with 20-s intervals.

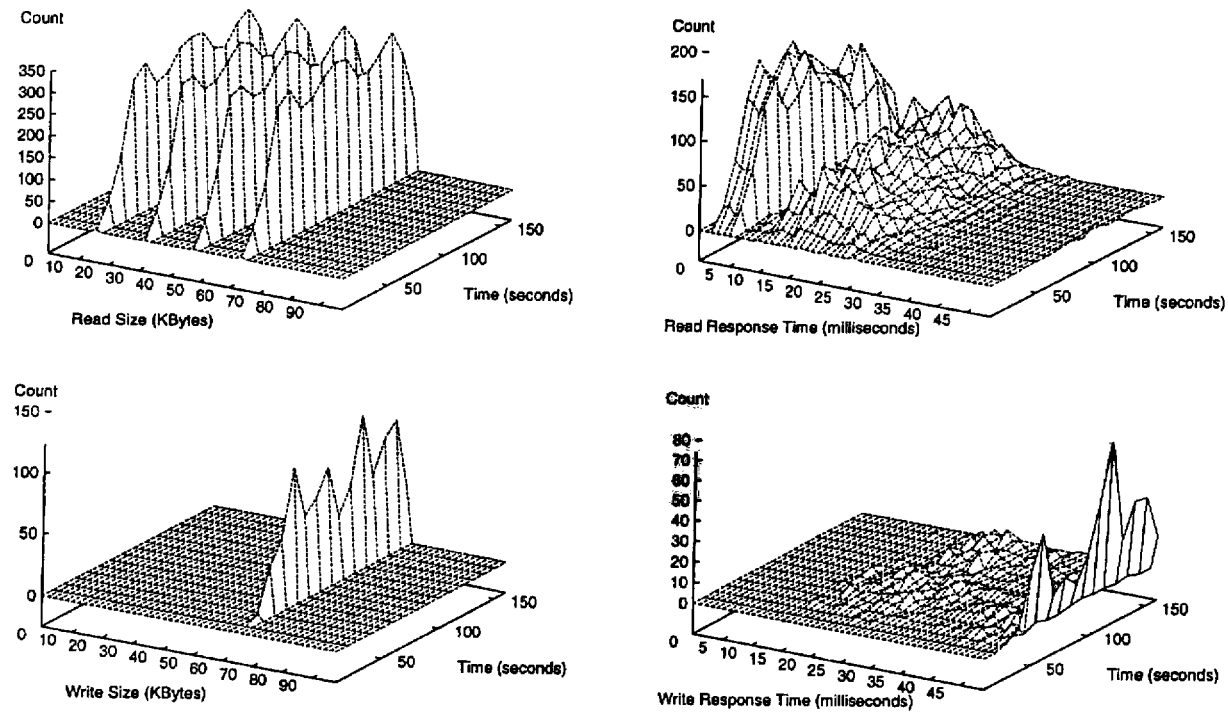


Fig. 2 Read benchmark physical I/O histograms (64 Seagate disks without buffering)

Several striking attributes are immediately apparent from the table and figures. First, both the number of physical operations and the total data volume differ markedly from that requested at the application level. For the write benchmark, PFS not only must update the file metadata as file blocks are written, it must also write the file system metadata to reflect the creation of new files.

When a processor creates a new file, other experiments (not displayed here due to space limitations) have shown that PFS must write two 64-KB blocks of metadata on each disk across which the file will be striped.[2] Hence, the physical data volume of Table 2 rises with the number of disks (e.g., when 64 processors create a file that is striped across 64 disks, over 512 MB of metadata must be written before any application data are stored).

Although this metadata overhead is partially an artifact of the interactions between PFS and OSF/1, it highlights the often hidden costs of scaling workstation file systems to hundreds or thousands of disks. The UNIX *inode* scheme was originally developed to efficiently support

"... both the number of physical operations and the total data volume differ markedly from that requested at the application level."

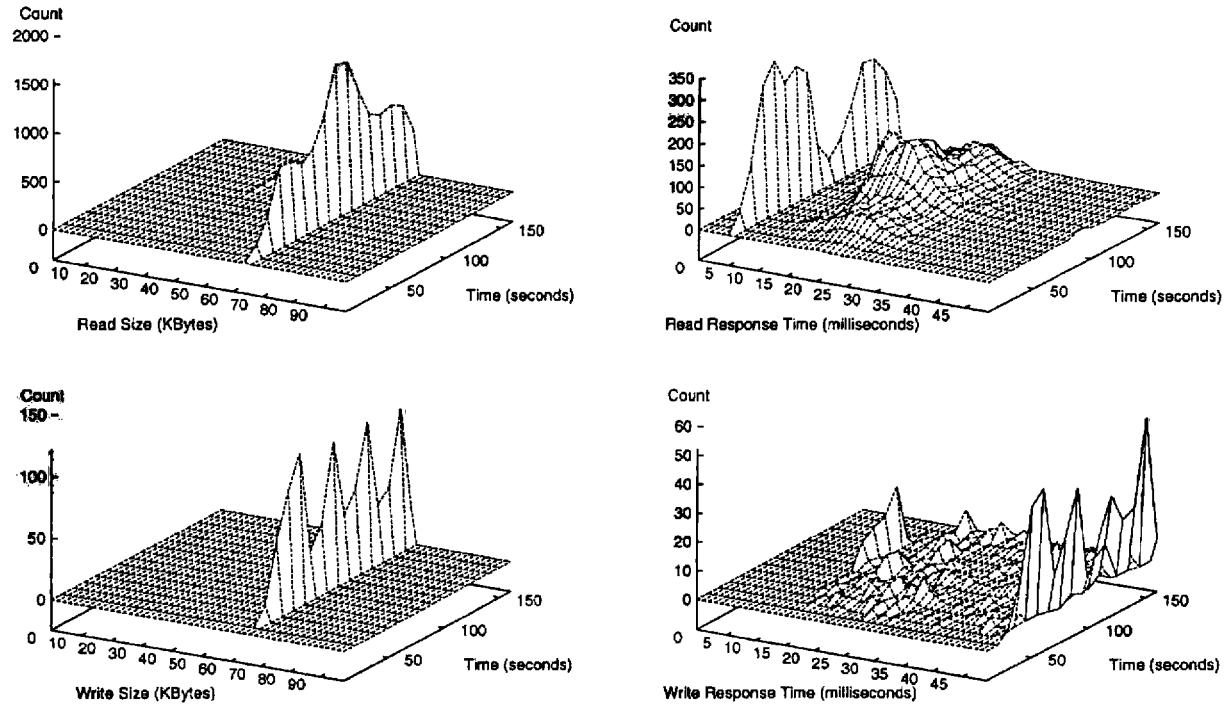


Fig. 3 Read benchmark physical I/O histograms (64 Seagate disks with buffering)

small files and dynamic growth. In contrast, files associated with scientific applications are both very small and very large. As others have also noted, this suggests that for the latter, extent-based allocation schemes could increase physical contiguity on storage devices and reduce metadata manipulation costs.

Table 2 and Figures 2 and 3 also show that the file-buffering policy has profound effects on physical request sizes and response times for both reads and writes. With the default PFS policy (no buffering), satisfying sequential 80-KB application read or write requests requires reading from or writing data to two disk stripes. Although buffering and write-behind ameliorate this overhead, they incur additional memory costs—a more flexible file system would match the size and placement of the disk stripes to the request size and access pattern.

Turning to the read benchmark, both Figures 2 and 3 illustrate the effects of SCSI command queuing and track buffering on the Seagate disks. Some read prefetches are

satisfied from the track buffers, whereas others see the full seek, rotation, and transfer latencies.

Figure 2 shows that all multiples of 16 KB are retrieved to satisfy unbuffered read requests. Again, this reflects the mismatch between request size and striping factor. A succession of 80-KB requests requires 16-, 32-, and 48-KB portions of the 64-KB stripes. With similar benchmarks using 16-KB request sizes, we observe that the application and physical request counts and volumes *were* identical for unbuffered I/O.

With buffering enabled, PFS and OSF/1 sequentially prefetch data in 64-KB units for file reads, yielding the single physical request size of Figure 3. This reduces the number of actual physical reads but increases the total I/O volume, as Table 2 illustrates. However, Figure 3 shows that even with a read interrequest interval of 1 s, the prefetching algorithm is unable to retrieve all data before they are requested. In turn, this results in the response time spike at the top of Figure 3.

Third, as Table 2 and Figures 2 and 3 show, the read benchmark generates a nontrivial number of physical writes. For reads, these writes accrue from metadata processing to record last access times for file blocks. Every 30 s, all I/O nodes write this data to their disks. By their nature, these file synchronizations are bursty, resulting in large disk response times. Moreover, comparison of Figures 2 and 3 shows that the interaction between prefetching and file synchronization adversely affects both.

Although read and write benchmarks highlight many possible interactions between access patterns, file system policies, and hardware configurations, their resource demands often are simpler and more regular than those in realistic applications. Hence, we turn now to an analysis of a large, I/O intensive parallel application.

5 Quantum Chemistry Code

As noted earlier, one of the primary goals of the Scalable I/O Initiative is analyzing the I/O patterns present in a large suite of scientific and engineering codes. These span a broad range of disciplines and have been the subject of several application characterization studies (Crandall et al., 1996; Smirni et al., 1996; Reed, Elford, Madhyastha, Scullin, et al., 1996).

As an initial basis for integrated analysis of application and physical I/O analysis, we selected one code (MESSKIT) from the Scalable I/O Initiative suite. This code has been the subject of earlier application analysis (Crandall et al., 1996) and is representative of the I/O patterns observed in parallel scientific applications. As such, it provides a baseline for comparison of logical and physical I/O patterns.

MESSKIT is a Fortran implementation of the Hartree-Fock self-consistent field method (High Performance Computational Chemistry Group, 1995) that computes the electron density around a molecule by considering each electron in the molecule in the collective field of the others. The implementation uses basis sets derived from the atoms and the relative geometry of the atomic centers. Atomic integrals are then calculated over these basis functions and are used to approximate molecular density. A Fock matrix is derived using molecular densities and the atomic integrals. Finally, a self-consistent field method is used until the molecular density converges to within an acceptable threshold. Because a Fock matrix of size N generates $O(N^2)$ one-electron and $O(N^4)$ two-electron integrals, the total I/O demand for realistic problems is beyond what can be feasibly supported with current parallel I/O systems.

The MESSKIT code consists of three distinct programs that operate as a logical pipeline, with each stage accepting input from the previous one.

- *psetup*: The processors read the initial files, transform the data in ways needed by the later phases, and write the result to disk.
- *pargos*: Each processor locally calculates and writes to disk integrals that involve either one or two electrons.
- *pscf*: Finally, each processor repeatedly reads its private integral files to retrieve the necessary quadrature data and solves the self-consistent field equations. The results are periodically collected and written to disk by processor zero.

Although both the *pargos* and *pscf* phases are I/O intensive, for brevity's sake, we consider only the input intensive *pscf* phase below.

6 Logical I/O Patterns

Table 3 shows the I/O behavior of MESSKIT's *pscf* phase, as captured using our application I/O characterization software.[3] The data were obtained by executing the code on 64 processors and different hardware configurations, all using a small 16-atom test problem and the Intel PFS `M_UNIX` file access mode, a direct extension of standard UNIX file system semantics.[4]

Table 3 shows that even for this small test problem, the *pscf* phase requires over 50,000 accesses to secondary storage to retrieve quadrature data. With the older, slower

Table 3
***pscf* Logical I/O Summary (64 processors)**

Operation	Operation Count	64 Disks		12 RAIDs	
		I/O Time (seconds)	Execution Time %	I/O Time (seconds)	Execution Time %
Open	93	32.42	0.08	29.99	0.05
`read`	51,313	4223.89	10.54	24,082.92	37.07
`seek`	429	2.94	0.01	3.97	0.01
`write`	207	4.10	0.01	5.84	0.01
`close`	92	15.11	0.04	11.99	0.02
All I/O	52,134	4278.47	10.67	24,134.71	37.15

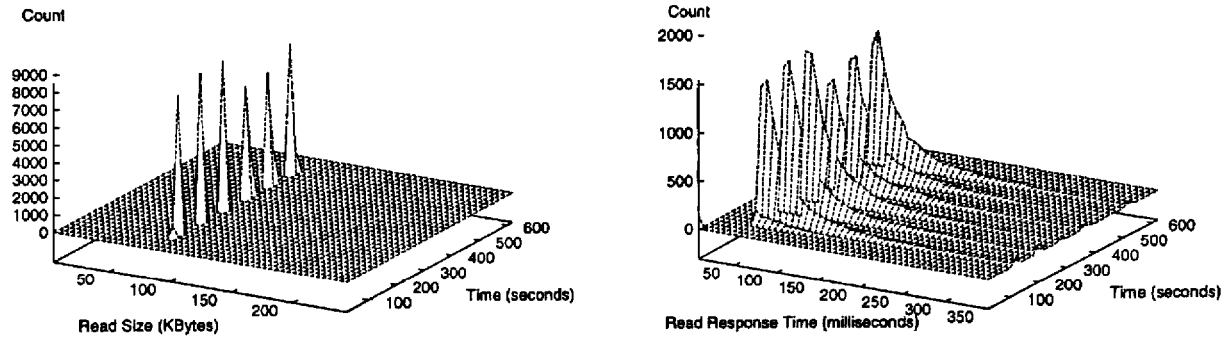


Fig. 4 ***pscf* logical I/O histograms (64 disks)**

RAID-3 disk array, these accesses consume nearly 40% of all execution time, although this decreases to nearly 10% with a larger number of faster disks.

In addition to input costs, Table 3 also illustrates the high cost of file open and close; each file open averages roughly 0.3 s per processor. As described in Section 4, these costs arise from PFS metadata manipulation. In principle, one could preallocate space for the output files, a method used by many database systems to reduce metadata manipulation overhead. However, for scientific applications like MESSKIT, the size of the output files is strongly dependent on the input data, and limited excess storage space is available for preallocation.

As a basis for comparison with physical I/O histograms, Figure 4 shows a histogram of application read sizes and durations for the 64-disk configuration. Clearly, the *pscf* read activity is bursty, with six cycles visible in

Table 4
***pscf* Logical/Physical I/O Comparison (64 processors)**

Operation	Logical		64 Disks		16 Disks		12 RAIDs	
	Count	Volume (MB)	Count	Volume (MB)	Count	Volume (MB)	Count	Volume (MB)
Unbuffered								
`read`	51,313	3992.47	257,228	4008.88	257,398	4009.01	264,844	4478.04
`write`	207	3.67	5071	242.79	2169	99.08	2121	85.08
`read + write`	51,520	3996.14	262,299	4251.67	259,567	4108.09	266,965	4563.12
Buffered								
`read`	51,313	3992.47	81,591	5045.37	116,842	7261.47	128,348	7949.03
`write`	207	3.67	4484	216.91	1602	72.10	1578	69.21
`read + write`	51,520	3996.14	86,075	5262.28	118,444	7333.57	129,926	8018.24

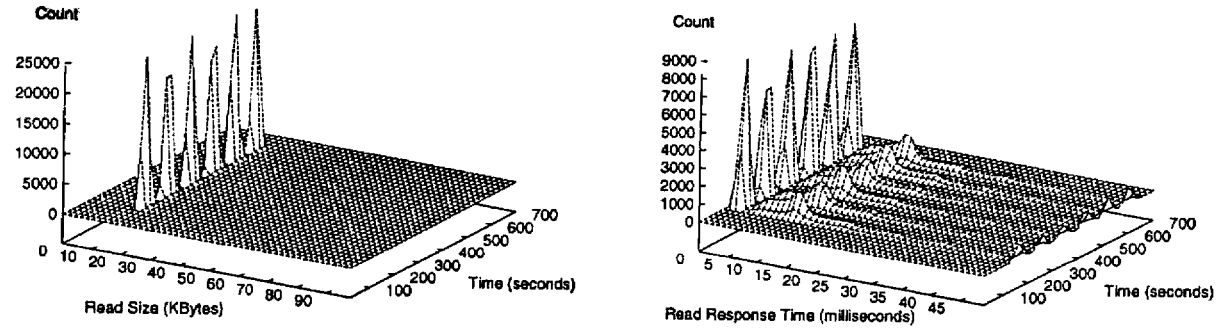
Figure 4. Most read request sizes are near 80 KB, although a few are near 200 KB. As a consequence of request burstiness, the application read durations are highly variable—PFS makes no attempt to minimize read latency by aggressively prefetching during compute intensive intervals.

7 Physical I/O Patterns

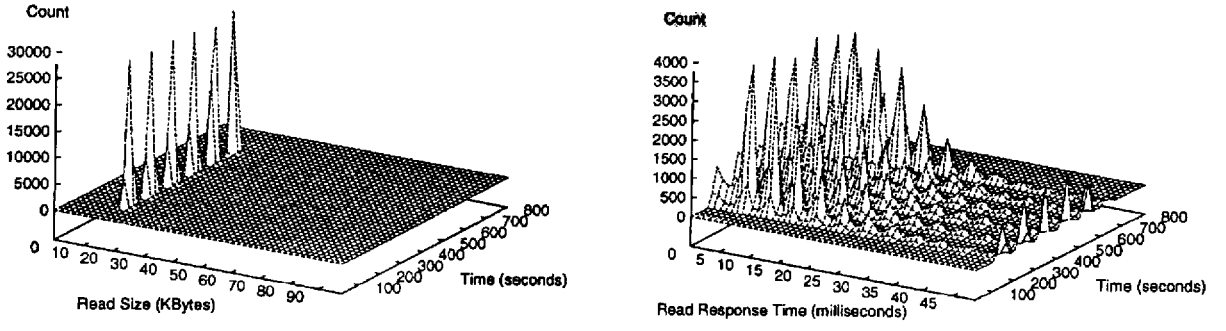
Using the SCSI device driver instrumentation described in Section 3, we measured the physical I/O characteristics for the MESSKIT code phases on each of our three SCSI disk configurations. Table 4 summarizes these measurements for the *pscf* phase.

When PFS buffering is disabled, the physical read data volume roughly equals the logical data volume, although the number of physical requests exceeds the number of logical requests by a factor of four. With buffering, the data volume for physical reads increases substantially, but the total number of physical requests declines, reflecting the fact that prefetching retrieves a smaller number of larger 64-KB stripes. As with the benchmarks of Section 4, the majority of the write traffic is attributable to metadata updates.

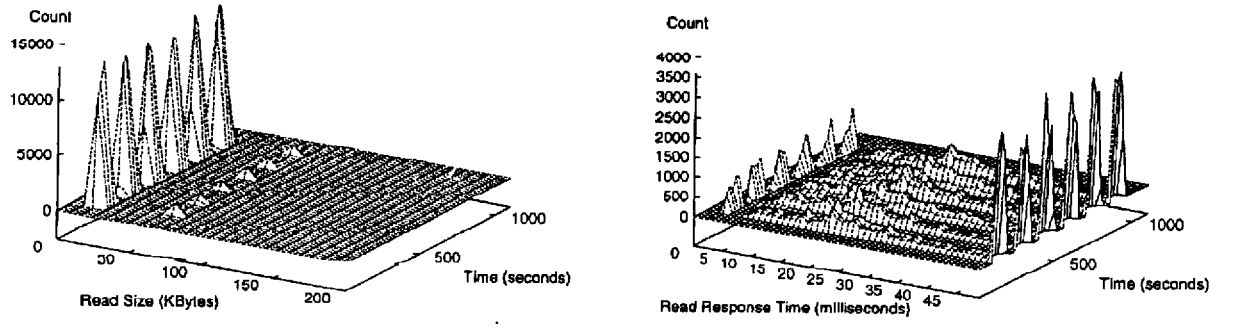
As a complement to the logical access histograms of Figure 4, Figure 5 illustrates the temporal distribution of physical request sizes and durations for the *pscf* phase with PFS file buffering disabled. The logical and physical access patterns are quite similar, although the 80-KB reads of Figure 4 become 16-KB reads in Figure 5.



(a) 64 Seagate Disks



(b) 16 Seagate Disks



(c) 12 RAIDs

Fig. 5 ***pscf*** **physical I/O histograms (64 processors without buffering). (a) 64 Seagate disks, (b) 16 Seagate disks, (c) 12 RAIDs**

More striking is the effect of changing hardware attributes on the distribution of physical request response times. The SCSI standard supports multiple outstanding requests through a mechanism called command queuing. Using this mechanism, a disk controller can resequence requests based on internal state to minimize request response times. The older RAID-3 disk arrays of Table 1 do not support command queuing, but the newer Seagate disks do.

In Figure 5, with 64 Seagate disks, the I/O system has a sufficient parallelism to avoid long queuing delays at each disk. This, together with command queuing and on-board request resequencing, allows the disks to satisfy most requests from the disk track buffers; these are the response times below 5 ms in Figure 5. As the number of disks declines to 16, a larger fraction of the requests require disk arm movement or encounter queuing delays. Finally, the 12, slower RAID-3 disks are saturated during application request bursts and lack command queuing to resequence requests. In consequence, most physical requests see large queuing delays.

7.1 QUEUING AND BLOCK RUNS

Although Figure 5 shows both the distribution of physical request sizes and the pernicious effects of insufficient hardware parallelism, it reveals little about the locality of physical requests or the sizes of disk queues. Using our SCSI driver instrumentation, we also captured SCSI block run lengths (i.e., the number of consecutive blocks accessed on each disk) and the distribution of queue sizes.

An analysis of these data shows that the product of the block run length and block size generally equals the size of the physical requests. This means that there is little or no locality across successive physical read requests, even though the high-level file operations are almost all sequential reads in the MESSKIT code.

PFS file striping distributes all files across all available disks in the file system. When multiple processes contend for access on a single disk, successive accesses to that disk have little locality. Simply put, the disks see nonsequential requests, necessitating disk arm movements and increasing access times.

Turning to device queue lengths, Figure 6 illustrates the temporal distribution of queue sizes for the 12 RAID, 3 disk arrays. As expected from the high response times of Figure 5, these disks are operating in saturation, with queue lengths exceeding 40 requests during periods of

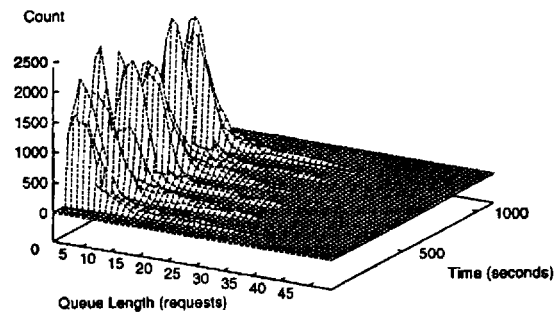


Fig. 6 ***pscf*** **driver queue (12 RAIDs)**

"When multiple processes contend for access on a single disk, successive accesses to that disk have little locality. Simply put, the disks see nonsequential requests, necessitating disk arm movements and increasing access times."

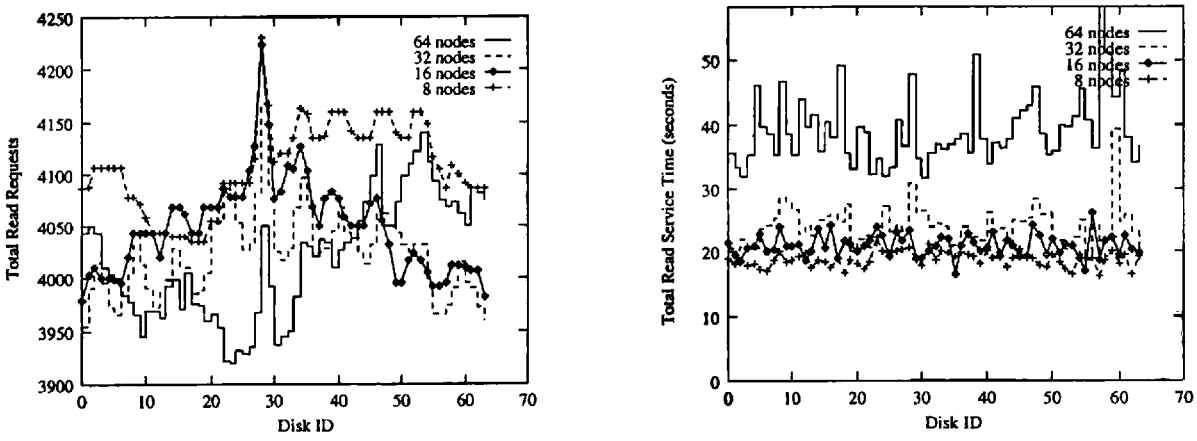


Fig. 7 ***pscf*** **disk request distributions (64 Seagate disks)**

high activity. Even the 16- and 64-disk configurations see maximum queue lengths of 10 and 20, respectively.

7.2 DISK LOAD DISTRIBUTIONS

As noted earlier, Intel's PFS stripes file data in 64-KB units, beginning with a randomly selected I/O node. For large sequential file accesses like those in MESSKIT, one would expect this data placement to distribute the number of physical requests nearly equally across the I/O nodes and disks. Surprisingly, this is not the case for the *pscf* phase.

Figure 7 shows both the number of requests and the total read service time for each disk in the 64-disk configuration as a function of application parallelism. When the number of application processors is no larger than the number of disks, read time is largely independent of application parallelism levels. However, when the number of contending processors equals or exceeds the number of disks, the total service time increases substantially.

Counterintuitively, smaller numbers of processors generate larger numbers of requests, even though the total application I/O volume in *pscf* is independent of the number of processors. This, together with the lack of physical locality described in Section 7.1, suggests that there are substantial opportunities for performance optimization within PFS.

8 Discussion and Implications

"SCSI device driver instrumentation and metric histograms strike the right balance between the limited detail provided by simple counts and the excessive overhead from tracing physical I/O within device drivers."

Based on our comparison of application I/O stimuli with physical I/O system responses on the Intel Paragon XP/S, several implications for both performance measurement toolkits and next-generation PFSs are clear. First, SCSI device driver instrumentation and metric histograms strike the right balance between the limited detail provided by simple counts and the excessive overhead from tracing physical I/O within device drivers.

In our experience, histogramming is efficient and provides a wealth of time-varying detail on request sizes, device driver service times, request run lengths, driver and device queue lengths, and interarrival times. When complemented by user-level measurement of application I/O patterns, one can analyze the interactions of application requests, PFS policies, and disk parallelism.

Second, maintaining metadata is a major overhead for file systems that naively retain UNIX file system semantics. Current teraflop systems are being configured with thousands of disks, and proposed petaflop systems would have nearly 100,000 disks. Straightforward extrapolation of PFS-style metadata storage for such systems would entail writing gigabytes of metadata just to create a file. Alternative representations (e.g., extent-based storage) are needed that are more amenable to file striping while retaining fault tolerance.

Third, both our benchmarks and the MESSKIT chemistry code illustrated the limitations of a single disk stripe size and distribution policy. When application requests are not a natural multiple of the stripe size, either because they are much smaller or much larger, the number and volume of physical I/O can differ markedly from that specified by the application. This mismatch, together with the demonstrable overheads, suggests that next-generation file systems must support flexible storage formats that can dynamically select a stripe size and a distribution of stripes across disks.

As a complement to more flexible data distributions, file system policies must aggressively exploit application access patterns. Intel PFS and OSF/1 include simple read-ahead and write-behind policies with LRU cache replacement. For the MESSKIT application, these policies fragment application requests, generate unnecessary disk activity, and fail to exploit bursty behavior to aggressively prefetch data during idle periods. For example, we observed that the combination of PFS policies and data distributions across disks eliminated almost all access locality present in the application pattern. Automatic access pattern classification (Madhyastha and Reed, 1996),

"Automatic access pattern classification, coupled with performance-directed adaptive control for policy selection, could dynamically tailor policies to access patterns."

coupled with performance-directed adaptive control for policy selection (Reed, Elford, Madhyastha, Smirni, et al., 1996), could dynamically tailor policies to access patterns.

Fourth, despite the temptation to sacrifice I/O systems for additional processors or primary memory, high performance parallel systems can realize their potential only when balanced. For the MESSKIT application, we saw that a 4:1 processor-to-disk ratio was insufficient to maximize performance. Although Amdahl's suggestion that an MIPS of computing must be balanced by a megabyte of memory and a megabyte per second of I/O may not hold for massively parallel systems, the premise of system balance remains true. Our experiments suggest that the number of disks in parallel I/O systems must be within a small constant factor of the number of processors for many scientific applications.[5]

Finally, characterization studies are by their nature inductive, covering only a small sample of the possibilities and attempting to extract more general patterns. Although the benchmarks and MESSKIT chemistry application we studied on the Intel Paragon XP/S are but a few samples from a large space of possible I/O patterns, earlier application characterization studies (Crandall et al., 1996; Smirni and Reed, 1996, 1997; Reed, Elford, Madhyastha, Scullin, et al., 1996; Purakayastha et al., 1995) suggest that our selections are representative of current practice. Although a wider range of experiments is desirable, the level of instrumentation and experiments we conducted required access to the operating system code and single-user time to load experimental operating system kernels. This restricted our ability to conduct comparative experiments on multiple platforms. A complete exploration will require analysis of additional applications, hardware platforms, and PFSs.

9 Conclusions and Futures

We have examined the interactions of application I/O requests, PFS policies, and disk hardware configurations using both application I/O measurements and SCSI device driver instrumentation. This analysis suggests that the physical I/O patterns induced by application requests are strongly affected by data-striping mechanisms, file system policies, and disk hardware attributes. Simply put, no single file policy or data distribution is optimal for all application access patterns.

Based on this analysis, we are exploring three approaches to I/O optimization: qualitative access pattern

classification based on trained neural networks and hidden Markov models (Madhyastha and Reed, 1996), flexible policy selection using fuzzy logic techniques (Reed, Elford, Madhyastha, Smirni, et al., 1996), and adaptive storage formats based on redundant representations.

ACKNOWLEDGMENTS

We thank Evgenia Smirni, Christopher Elford, Tara Madhyastha, and Ruth Aydt for their insights on parallel I/O and instrumentation. We also thank Rick Kendall of the Molecular Science Software Group at the Molecular Science Research Center, Pacific Northwest National Laboratory, for the MESSKIT code. All data presented here were obtained from code executions at the Caltech Center for Advanced Computing Research. This work was supported in part by the Defense Advanced Research Projects Agency under DARPA Contracts DABT63-94-C0049 (Scalable I/O Initiative), DAVT63-91-C-0029, and DABT63-93-C0040, by the National Science Foundation under Grant NSF ASC 92-12369, by a joint Grand Challenge grant with Caltech, and by the National Aeronautics and Space Administration under NASA Contract NAG-1-613.

BIOGRAPHIES

Huseyin Simitci is a doctoral candidate in computer science at the University of Illinois at Urbana-Champaign. His research interests include parallel file systems, high performance computing, and intelligent software. He obtained an M.S. and a B.S. from Bilkent University, Ankara, Turkey, in 1994 and 1992, respectively. He is a member of the Pablo Research Group.

Daniel A. Reed is a professor and head of the Department of Computer Science at the University of Illinois at Urbana-Champaign. In addition, he holds a joint appointment as a senior research scientist with the National Center for Supercomputing Applications. He received a B.S. in computer science from the University of Missouri at Rolla in 1978 and an M.S. and Ph.D., also in computer science, from Purdue University in 1980 and 1983, respectively. He was a recipient of the 1987 National Science Foundation Presidential Young Investigator Award.

NOTES

1. As a basis for comparison, this combination of request size and access pattern was chosen to match the application access pattern in Section 5.

2. Traces showed that in most cases there are two block writes per stripe file, but occasionally the number of blocks is one or three; thus, the small variations in the metafile volume occur.

3. In Table 3, the I/O time column is the sum of all time spent performing I/O across all processors.

4. The PFS `M_ASYNC` mode, which does not preserve file access atomicity when files are concurrently opened by multiple processes, is a potentially lower overhead alternative to the `M_UNIX` mode. However, in the MESSKIT code, each file is accessed by a single processor.

5. Clearly, for some application domains this is not the case.

REFERENCES

Baker, M. G. 1991. Measurements of a distributed file system. *Proceedings of the Thirteenth Symposium on Operating System Principles 25*. Association for Computing Machinery, pp. 198-212.

Crandall, P., Aydt, R. A., Chien, A. A., and Reed, D. A. 1996. I/O characterization of scalable parallel applications. In *Proceedings of Supercomputing 1995*, San Diego.

Forin, A., Golub, D., and Bershad, B. 1991. An I/O system for Mach 3.0. In *Proceedings of the USENIX Mach Symposium*, USENIX, pp. 163-176.

Gibson, G. A., Vitter, J. S., and Wilkes, J. 1996. Strategic directions in computing research: Working group on storage I/O issues in large-scale computing. *ACM Computing Surveys* 28(4): 779-793.

High Performance Computational Chemistry Group. 1995. NWChem, a computational chemistry package for parallel computers, version 1.1. Available at Pacific Northwest National Laboratory, Richland, WA, 99352, U.S.A..

Jensen, D. W., and Reed, D. A. 1993. File archive activity in a supercomputing environment. In *Proceedings of the 1993 ACM International Conference on Supercomputing*, July.

Kotz, D., and Nieuwejaar, N. 1994. Dynamic file-access characteristics of a production parallel scientific workload. In *Proceedings of Supercomputing '94*, Los Alamitos, November, pp. 640-649.

Lawrie, D. H., Randal, J. M., and Barton, R. R. 1982. Experiments with automatic file migration. *IEEE Computer*, July, pp. 45-55.

Madhyastha, T., and Reed, D. A. 1996. Intelligent, adaptive file system policy selection. In *Proceedings of Frontiers '96*, October, 172-179.

Miller, E. L., and Katz, R. H. 1991. I/O behavior of supercomputer applications. In *Proceedings of Supercomputing '91*, November, pp. 567-576.

Pasquale, B. K., and Polyzos, G. A. 1993. Static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of Supercomputing '93*, Portland, November, pp. 388-397.

Pasquale, B. K., and Polyzos, G. C. 1994. Dynamic I/O characterization of I/O intensive scientific applications. In *Proceedings of Supercomputing '94*, Washington, DC, November, pp. 660-669.

Pool, J. T. 1996. Scalable I/O Initiative. California Institute of Technology. Available at http://www.ccsf.caltech.edu/SIO/

Purakayastha, A., Ellis, C. S., Kotz, D., Nieuwejaar, N., and Best, M. 1995. Characterizing parallel file access patterns on a large-scale multiprocessor. In *Proceedings of the Ninth International Parallel Processing Symposium*, April, pp. 165-172.

Reed, D. A., Aydt, R. A., Noe, R. J., Roth, P. C., Shields, K. A., Schwartz, B. W., and Tavera, L. F. 1993. Scalable perfor-

mance analysis: The Pablo performance analysis environment. In *Proceedings of the Scalable Parallel Libraries Conference*, edited by A. Skjellum. Los Alamitos, CA: IEEE Computer Society Press, 1993, pp. 104-113.

Reed, D. A., Elford, C. L., Madhyastha, T., Scullin, W. H., Aydt, R. A., and Smirni, E. 1996. I/O, performance analysis, and performance data immersion. In *Proceedings of MASCOTS '96*, San Jose, February, pp. 1-12.

Reed, D. A., Elford, C. L., Madhyastha, T., Smirni, E., and Lamm, S. L. 1996. The next frontier: Interactive and closed loop performance steering. In *Proceedings of the 1996 International Conference on Parallel Processing Workshop*, Bloomington, August, pp. 20-31.

Ruemmler, C., and Wilkes, J. 1993. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Conference*, USENIX, pp. 405-420.

Ruemmler, C., and Wilkes, J. 1994. An introduction to disk drive modeling. *Computer* 27(3):17-28.

Seltzer, M., Chen, P., and Ousterhout, J. 1990. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX Conference*, January, USENIX, pp. 313-324.

Smirni, E., Aydt, R. A., Chien, A. A., and Reed, D. A. 1996. I/O requirements of scientific applications: An evolutionary view. *High Performance Distributed Computing*, pp. 49-59.

Smirni, E., Aydt, R. A., Chien, A. A., and Reed, D. A. 1996. I/O requirements of scientific applications: An evolutionary view. In *Proceedings of the Fifth IEEE International Symposium on High-Performance Distributed Computing*, August, pp. 49-59.

Smirni, E., and Reed, D. A. 1997. Workload characterization of I/O intensive parallel applications. In *Proceedings of the 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, June.

Smith, A. J. 1981. Analysis of long term file reference patterns for application to file migration algorithms. *IEEE Transactions on Software Engineering SE-7* 4:403-417.

Stritter, T. R. 1977. File migration. Ph.D. thesis, Department of Computer Science, Stanford University.

Wilkes, J., Worthington, B. L., Ganger, G. R., and Patt, Y. N. 1995. On-line extraction of SCSI disk drive parameters. In *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, Ottawa, Canada, May, pp. 146-156.